Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Dátové štruktúry a algoritmy

Zadanie 2 - Vyhľadávanie v dynamických množinách

Jakub Hrnčár
AIS ID: 110 800, ak. rok 2020/2021

1. Úvod do zadania

Úlohou zadania bolo porovnať viac druhov dátových množín, konkrétne rôzne implementácie stromov a hash tabuliek. Za úlohu bolo naprogramovať jeden strom a jednu tabuľku a ďalší jeden strom a jednu tabuľku prevziať z internetu. Ja som si vybral naprogramovať AVL strom a hash tabuľku s použitím metódy “chaining” a prevzal som RB strom a hash tabuľku s metódou “linear-probing”.

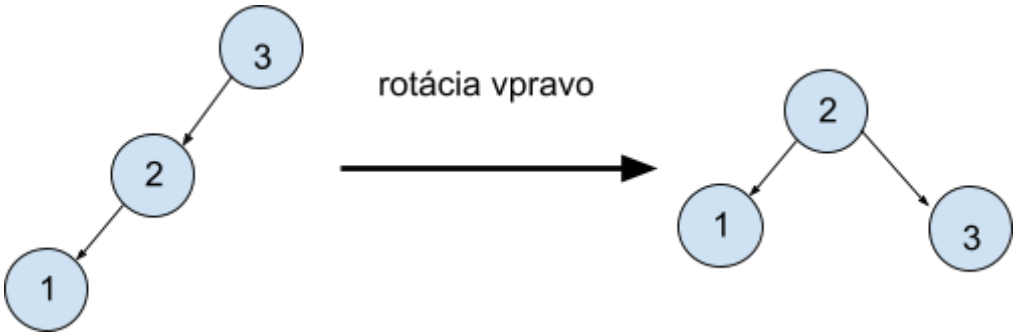
Do množín som vkladal vždy tie isté prvky a porovnával čas potrebný na spracovanie dát. V každej dátovej množine sme mali implementovať 2 funkcie: insert a search. Insert je funkcia na vloženie prvku do množiny a search prvok nájde, pričom ak ho nenájde tak vráti NULL.

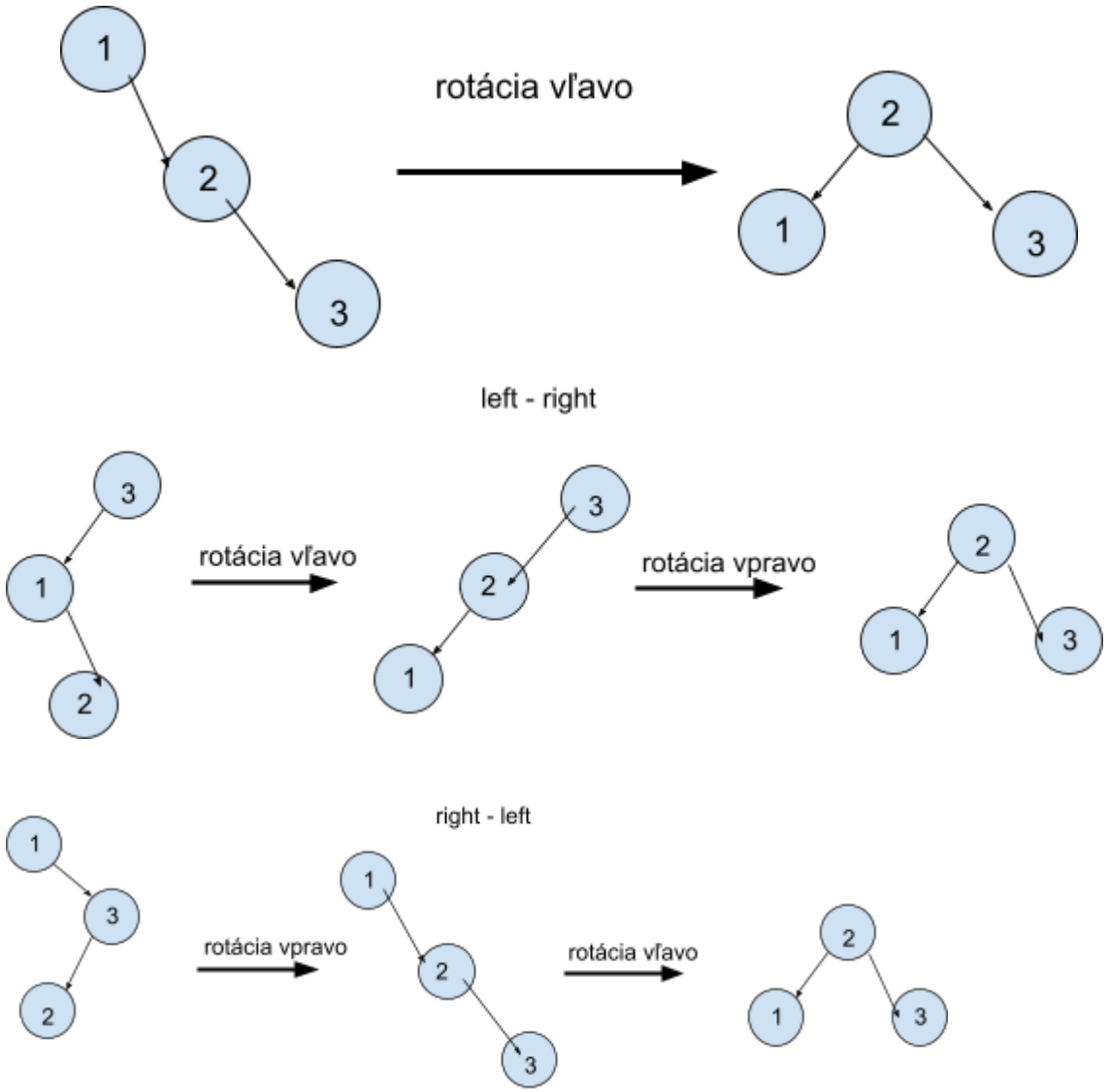
2. Stručný opis algoritmu

1. **AVL strom:**

   AVL strom je verzia binárneho vyhľadávacieho stromu. Z neho si preberá logiku vkladania: menší prvok od prvku nad ním ide vľavo, väčší ide vpravo. K tejto logike dodáva zaznamenávanie výšky, ktorá je vždy o 1 väčšia ako ľavý alebo pravý podstrom (podľa toho, ktorý je vyšší). Asi najpodstatnejší doplnok je tzv. “balance faktor”, ktorý nie je nič iné ako rozdiel výšok podstromov. Ak je tento faktor iné číslo ako -1,0 alebo 1, musia sa vykonať rotácie.

   Rotáciou označujeme algoritmus, ktorý vyváži AVL strom; presunie prvky zo strany kde ich je veľa do strany, kde ich je málo. Rotácia prebieha buď na celom strome alebo len na jeho časti. Poznáme 4 druhy rotácií: ľavá, pravá, ľavá-pravá a pravá-ľavá.





Modrá gulička znázorňuje 1 uzol a šípky znázorňujú prepojenia stromov. Dizajn uzlu je nasledovný:

- Hodnota - číslo, ktoré je vložené do stromu
- Výška - o 1 vyššia ako jej podstrom
- ľavý podstrom - ďalší uzol s takým istým dizajnom
- pravý podstrom - ďalší uzol s takým istým dizajnom

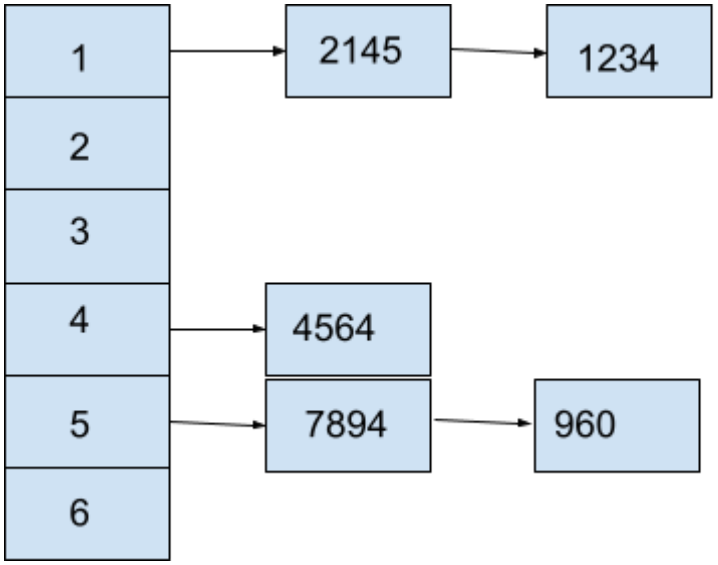
V jazyku C je uzol reprezentovaný štruktúra a je vložený do spájaného zoznamu, ktorý predstavuje celý strom:

```
typedef struct uzol{
    long long int  vyska;
    long long int  hodnota;
    struct uzol *vlavo;
    struct uzol *vpravo;
}UZOL;
```

2. Hash tabuľka - chaining

Druhá dátová množina, ktorú som sám vytvoril bola hash tabuľka. Keďže sa vkladajú len čísla, tak hashovacia funkcia je len modulo veľkosťou tabuľky. V tomto druhu dátovej množiny je nutné riešiť tzv. kolízie, ktoré nastanú, ak hashovacia funkcia vráti na 2 rôzne čísla ten istý výstup, napríklad: 3 mod 2 je to isté ako 5 mod 2 a teda 1, teda aj 3 aj 5 by sa chceli priradiť na index 1 a vznikne kolízia.

Ja kolízie riešim metódou chaining, ktorá pristupuje ku každému prvku tabuľky ako ku spájanému zoznamu a v prípade kolízie vloží nový prvok na začiatok spájaného zoznamu:



Jeden prvok tabuľky je preto v jazyku C reprezentovaný ako štruktúra:

```
typedef struct prvok{
    long long int kluc;
    struct prvok *dalsi;
}PRVOK;
```

Vkladanie do tabuľky je jednoduché, číslo na vstupe sa vloží do hashovacej funkcie, ktorá vráti index na ktorý sa má vložiť. Vyhľadávanie je taktiež jednoduché, použijem ten istý hashovací algoritmus na zistenie indexu a ak sa na tom indexe dané číslo nachádza, vrátim ho a ináč vrátim 0.

Ďalšia vlastnosť hashovacej tabuľky je to, že nemusí byť nekonečná. Ak chceme vytvoriť nekonečnú tabuľku, musíme pri istom naplnení tabulky ju zväčšiť. To prebieha nasledovným spôsobom: vytvorenie tabuľky s novou veľkosťou, vybratie čísla zo starej tabuľky, vloženie čísla do novej hashovacej funkcie unikátnej pre novú tabuľku, vloženie čísla do novej tabuľky na základe indexu z hashovacej funkcie.

Avšak, toto zväčšovanie nie je nutné - ak sa použije dostatočná štartovacia veľkosť tabuľky, môže sa vynechať. Preto som vytvoril 2 verzie tabuľky, ktoré aj porovnávam: jedna ktorá má štartovaciu veľkosť 50 prvkov a

zväčšuje sa o dvojnásobok pri naplnení na 50% a druhá, ktorá má vždy rovnakú veľkosť ako je počet vygenerovaných a vkladaných prvkov.

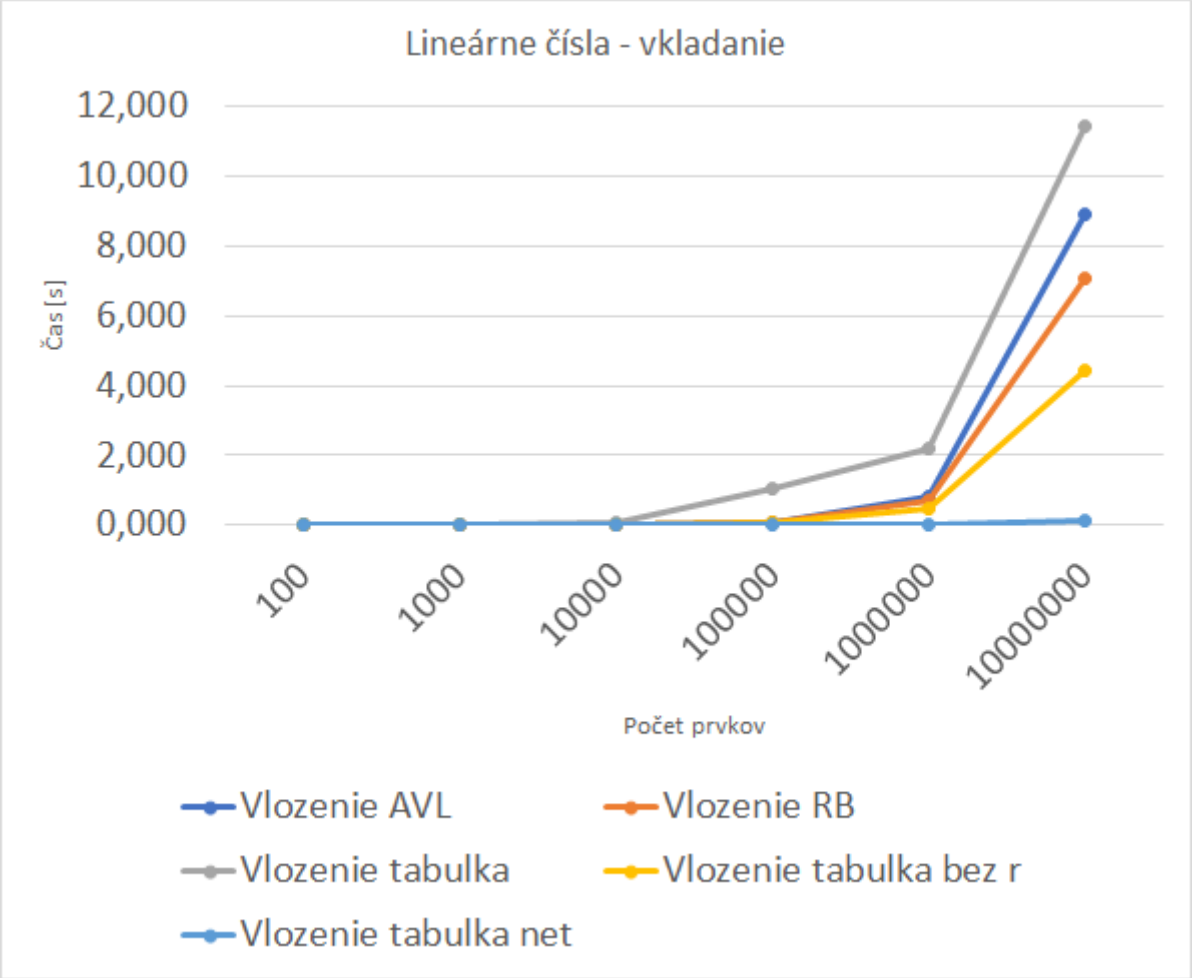
3. **Prevzatá implementácia - RB strom**
   Zdroj: https://gist.github.com/aagontuk/38b4070911391dd2806f
   Ako prevzatý strom som zvolil Red-Black strom, ktorý vytvoril Ashfaqur Rahman. Red-Black strom sa riadi týmito princípmi:
   - Koreň stromu je vždy čierny
   - Listy neobsahujú dáta a musia byť čierne
   - Čierna výška stromu udáva počet čiernych vrcholov od koreňa k listom
   - Červený vrchol má čierne deti a nemôžu byť 2 červené vrcholy za sebou
   - Maximálna dĺžka cesty k listu môže byť len dvojnásobkom dĺžky najkratšej cesty

   Môžeme rovnako používať rotácie, ako pri AVL strome.
   Pri insert a search sa využívajú tieto vlastnosti.
4. **Prevzatá implementácia - Hash tabuľka s linear-probing**
   Zdroj: https://www.thecrazyprogrammer.com/2017/06/hashing.html
   Prevzatá tabuľka pochádza z tejto edukačnej stránky. Programátor tu zvolil metódu riešenia kolízií ‘open-adressing’ a to konkrétne ‘linear-probing’. V praxi to znamená, že v prípade kolízie sa bude prechádzať tabuľkou za radom, až pokým sa nenájde najbližšie voľné miesto. Hashovacia funkcia a aj funkcia search zostávajú v podstate nezmenené. Taktiež tabuľka nemá zväčšovanie.

3. Testovanie programu

Testovanie programu je pomerne rozsiahle. Vytvoril som k testovaniu jeden veľký testovač, ktorý najprv vytvorí vzorku čísel a následne ich vloží do jednotlivých dátových množín a zmeria čas, ktorý bol potrebný na vykonanie operácie. V testovači sa dá vybrať, či vkladané čísla budú náhodne, alebo budú z aritmetickej postupnosti. Taktiež si v hlavičkovom súbore pocet.h vieme určiť počet vygenerovaných čísel.

Testy boli vykonávané na exponenciálne sa zvyšujúcom počte prvkov, a to: 100, 1 000, 10 000, 100 000, 1 000 000 a 10 000 000 prvkov, pričom tieto testy boli vykonané pri náhodných číslach a aj pri aritmetickej postupnosti čísel. Vkladať oba druhy čísel je dôležité, pretože pri pri niektorých množinách som takto schopný sa priblížiť k najhoršiemu a aj najlepšiemu prípadu. Program následne po vykonaní testu vloží zistené časy do súboru, kde som ich spracoval do podrobných grafov:

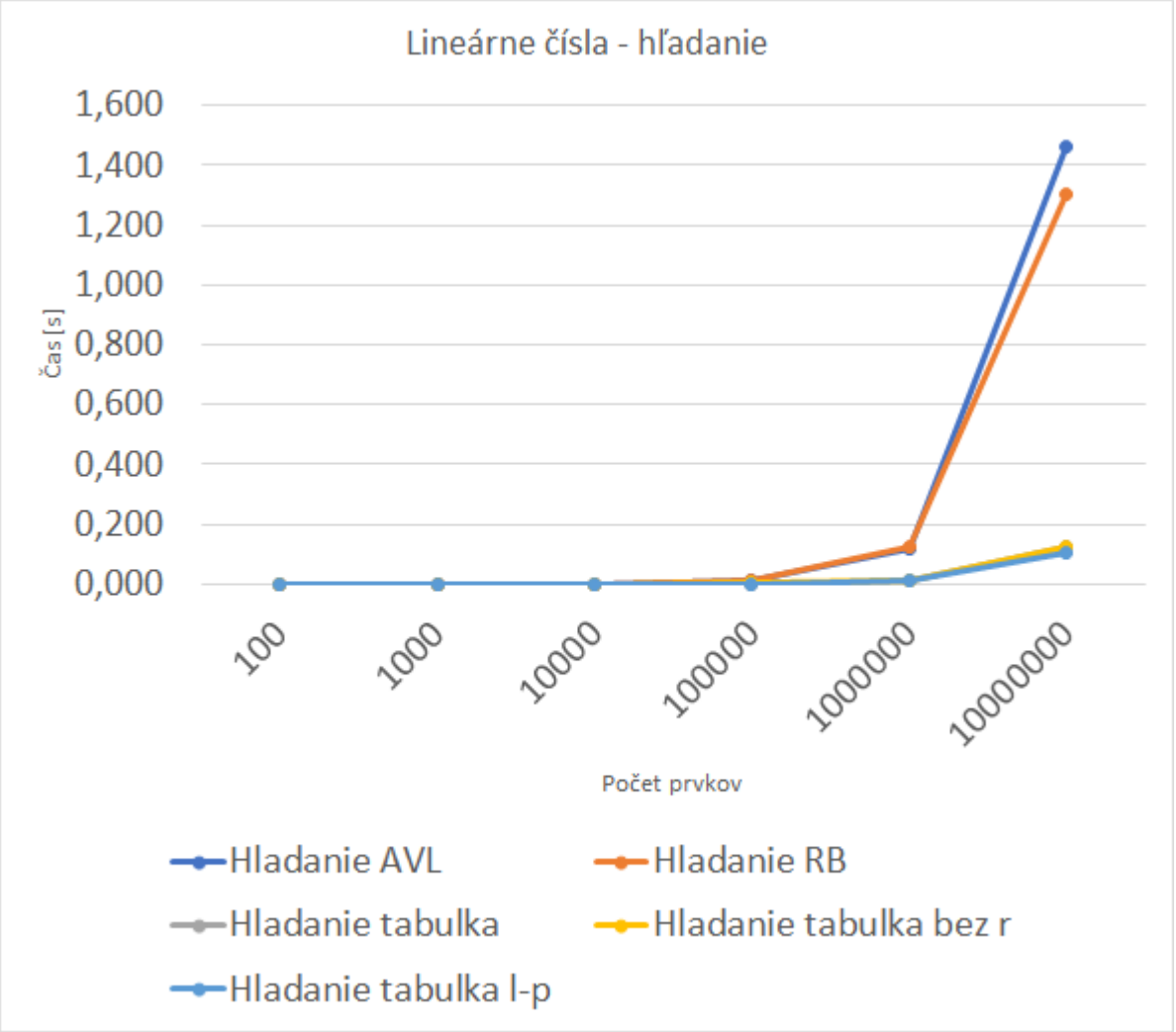


Jasný víťaz v tomto teste je prevzatá tabuľka s linear-probing. Je to kvôli tomu, že netreba vykonávať žiadne zväčšovanie a ani žiadny cyklus, nevznikajú žiadne kolízie.

Rozdiel oproti tabuľke bez zväčšenia je ten, že tabuľka bez zväčšenia musí byť špeciálne naformátovaná od začiatku, a to formátovanie trvá nejaký čas.

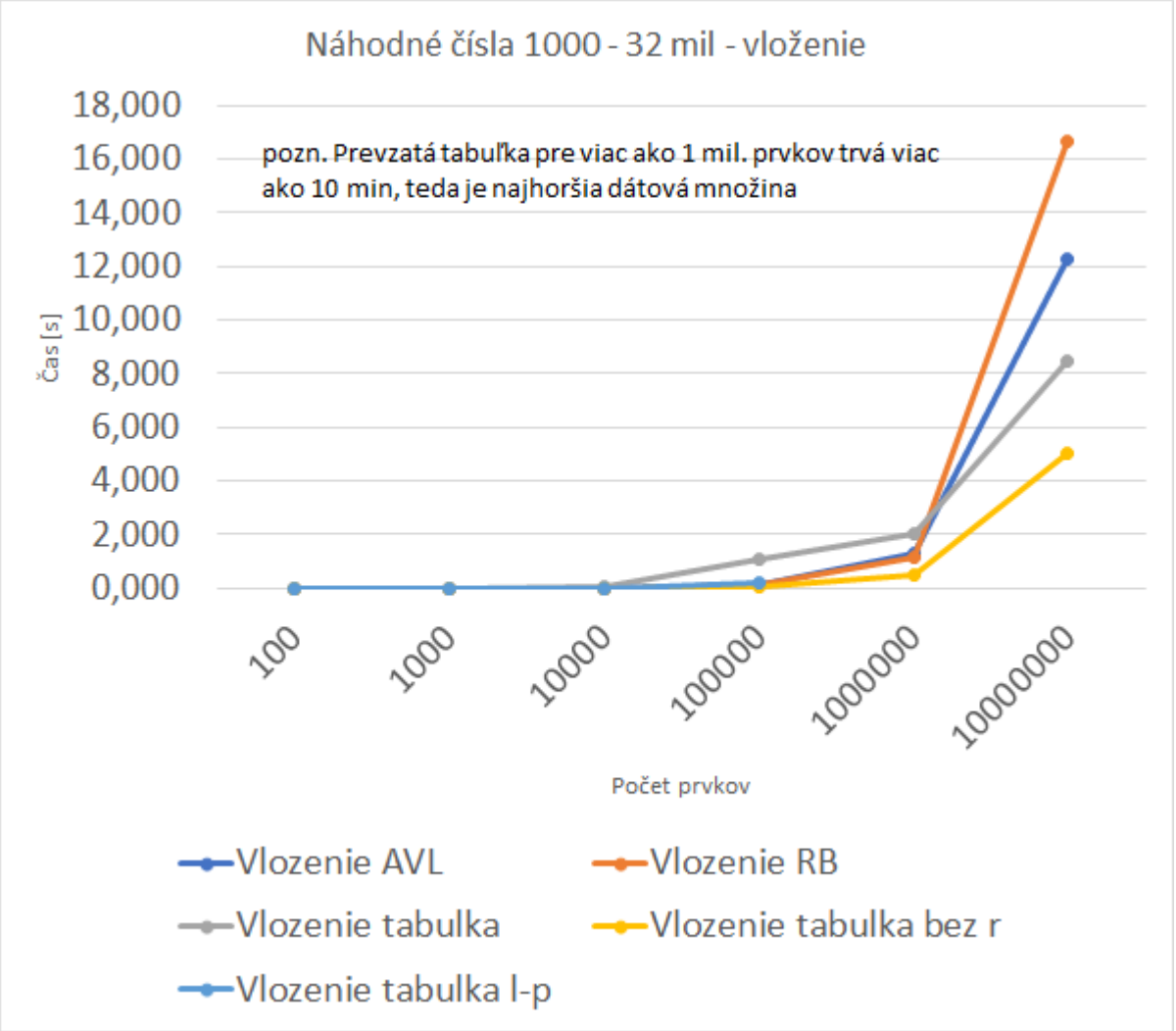
Najhoršie na tom je moja tabuľka, pretože síce nevzniká veľké množstvo kolízií, avšak tabuľka sa musí neustále zväčšovať, čo trvá veľmi dlho.

Stromy trvajú približne rovnako, avšak RB strom je o niečo lepší, pretože má menej strmú krivku. Spôsobené to je menším počtom rotácií.



Vo vyhľadávaní vidíme jasnú dominanciu všetkých druhov tabuliek, pretože ich vyhľadávanie má časovú zložitosť O(1). Nevznikajú žiadne kolízie, preto sa len pozrú a vrátia prvok.

Narozdiel od stromov, ktoré musia prechádzať na základe svojich vnútorných pravidiel niekedy aj celý strom.

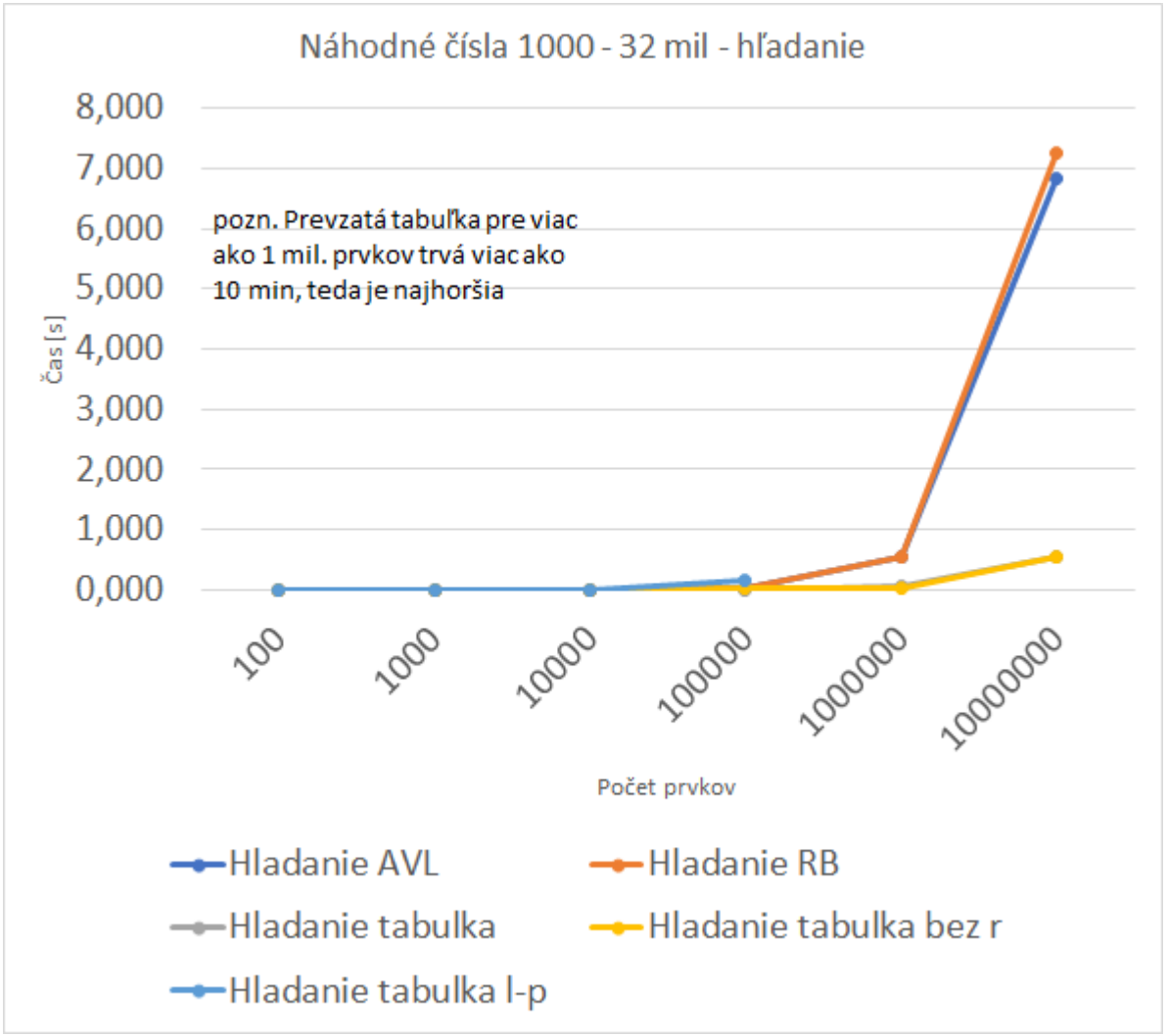


Pri teste s vkladaním náhodných čísel vidíme veľmi rozdielne výsledky.

Tabuľka s linear-probing vyhrávala, avšak akonáhle sa počet čísel začne zväčšovať a začne pribúdať kolízí, algoritmus to prestal zvládať a už pri milióne prvkov trvá vkladanie viac ako 10 minút, čo je nerealistický čas pre testovanie.

Najlepšie tento test zvládla tabuľka bez zväčšovania s algoritmom chaining, pretože kolízie sa riešili rýchlo a nebola zdržaná zväčšovaním. Tento rozdiel vidíme keď sa pozrieme na dĺžku trvania tabuľky so zväčšovaním. Ako aj vidíme na sklone oboch kriviek, sú vodorovné, čo len potvrdzuje predošlé tvrdenie.

Narozdiel od aritmetickej postupnosti čísel , kde vyhral RB strom, je zo stromov rýchlejší AVL strom, ktorý má aj menej strmú krivku, teda predpokladám, že bude rýchlejší aj pri ešte väčšom počte čísel.



Opäť, prebratá tabuľka bola vylúčená z tohto testu, pretože trvala extrémne dlho.

Pri hľadaní vidíme rozdiel iba v tom, že všetko trvá o niečo dlhšie. Stromy stále trvajú omnoho dlhšie ako tabuľky, aj napriek tomu, že tabuľky musia riešiť kolízie.

4. Záver

Na záver tu je zhodnotenie všeobecnej rýchlosti všetkých dátových množín, kde sa spočítavajú dosiahnuté rýchlosti insert aj search pri oboch typoch čísel:

1. Tabuľka bez zväčšenia - 0,47 skóre - pokiaľ vieme určiť maximálny počet vložených čísel, táto verzia hashovacej tabuľky je suverénny víťaz.
2. Tabuľka - 1,13 skóre - pokiaľ ale nevieme určiť veľkosť, na všeobecné používanie je táto hashovacia tabuľka najlepšie riešenie na uchovávanie čísel v dátovej množine.
3. AVL strom - 1,35 skóre - vo všeobecnom prípade je tento AVL strom rýchlejší ako tento konkrétny RB strom, avšak môže to byť pri iných implementáciách (ktoré sú viac optimalizované) opačne.
4. RB strom - 1,46 skóre - táto verzia RB stromu nedokázala poraziť AVL strom, avšak je pravdepodobné, že v nej existujú chyby, ktoré celý algoritmus spomaľujú
5. Tabuľka s linear-probing - suverénne najhoršie skóre - linear-probing je veľmi rýchla a jednoducho implementovateľná metóda, pretože tabuľka si nevyžaduje žiadnu prácu s adresami a žiadne špeciálne formátovanie. Avšak, aj keby test navrhnem tak, aby aj ostatné tabuľky nemali merané vytváranie, nič by to nezmenilo na tom, že pri veľkom množstve kolízií a pri veľkom počte čísel je táto implementácia takmer nepoužiteľná.